EECS 373 *Midterm 2*

Winter 2021

Name: ____Solutions_____ unique name: _____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

Exam Guidelines:

1. Part 1 of this exam will be given as an "online assignment" via Gradescope from 6:30pm-8:30pm on Friday 3/26.
2. Due to the online nature of the exam, this will be an "open book/notes" and "open web" exam. NO collaboration and No use of Code Emulators or IDEs is acceptable.
3. Multiple choice and short answer questions will be given and answered directly in Gradescope.
4. For questions that require handwritten answers students should write their answers on a blank piece of paper, take a photo and upload it to the "select files" field for that question. Students that prefer to use a tablet or other digital means to create their answer are welcome to do so.
5. Make sure to **save** your work as you go. You can **re-submit** your Gradescope exam as many times as you need during the allotted time.
6. If you need help or have questions, contact the instructors via zoom. Students are encouraged to log into Zoom at 6:20 pm for general instructions and announcements before the exam starts promptly at 6:30 pm. Students are not required to stay on zoom: https://umich.zoom.us/j/92181181546
7. Any announcements, clarifications or adjustments to the exam will be made on Piazza.
8. Part one of the exam is 14 pages including this one.
9. Don't spend too much time on any one problem.

Show your work and explain what you've done when asked to do so. State any assumptions.

Question 1: Multiple Choice - [20 points, 2 each]

1.1) What does ISR stand for?

- Interrupt Standard Routine
- Interrupt Service Routine
- Interrupt Software Routine
- Interrupt Synchronous Routine

1.2) What statement is true about ABI compliant functions?

- The function must not be interrupted
- The function can not call another function
- The function can use R0-R3 but must preserve their initial value
- The function must pass variables through the stack
- The function can return variables to R0-R3

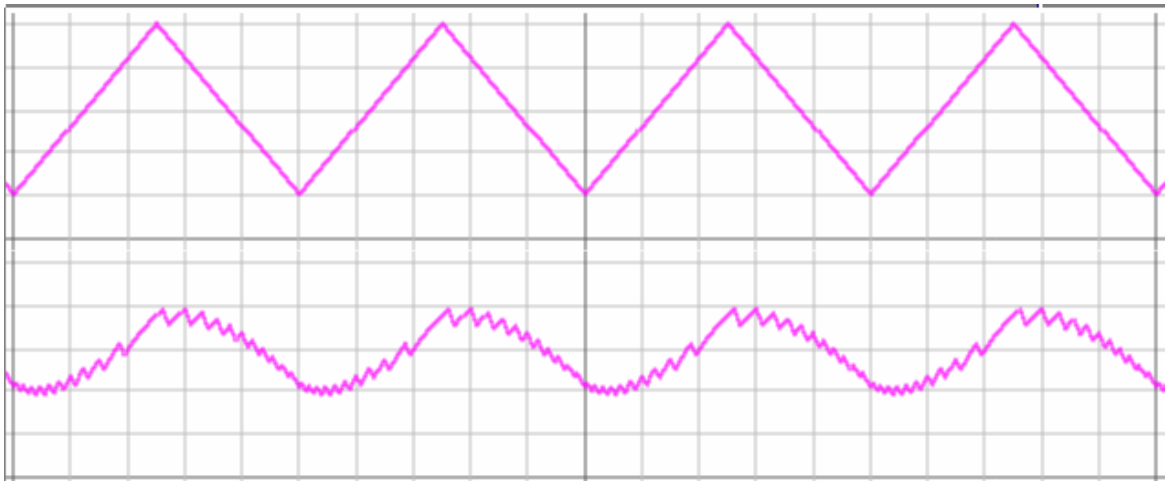
1.3) Which of the following is asynchronous to the operation of the core?

- Interrupts
- MMIO
- Polling
- Memory Access
- Hardware Multiplication

1.4) On consumer electronics devices, users can often perform a “Hard Reset” by holding down the power button for a set amount of time, which will cause the device to completely power off (so it can then be powered on and rebooted). Which type of timer would be a good choice to keep track of this time interval?

- Watchdog timer
- Digital Timer
- Capture Timer
- Compare Timer
- Auto Reload Timer

1.5) The following diagram shows a time series plot of two waveforms. The top shows the desired DAC output which is a 100 Hz triangle wave. The bottom shows the actual output of the DAC. Which type of DAC was most likely used to generate this output?



- Low Pass Filter Pulse-Width Modulation DAC
- Resistance Voltage Divider DAC (Aka Switch Tree DAC)
- R/2R Ladder DAC
- Binary-Weighted Resistor DAC
- FLASH DAC

1.6) Which ADC types must have a Sample and Hold Circuit to function properly?

- Flash
- Slope Integrator
- SAR
- Flash & Slope
- Slope & SAR
- SAR & Flash

1.7) Which Analog to Digital converter is considered to be simplest, fastest and most expensive?

- PWM Filter ADC
- Slope Integrator ADC
- Flash ADC
- Successive Approximation Register ADC
- Binary Weight ADC

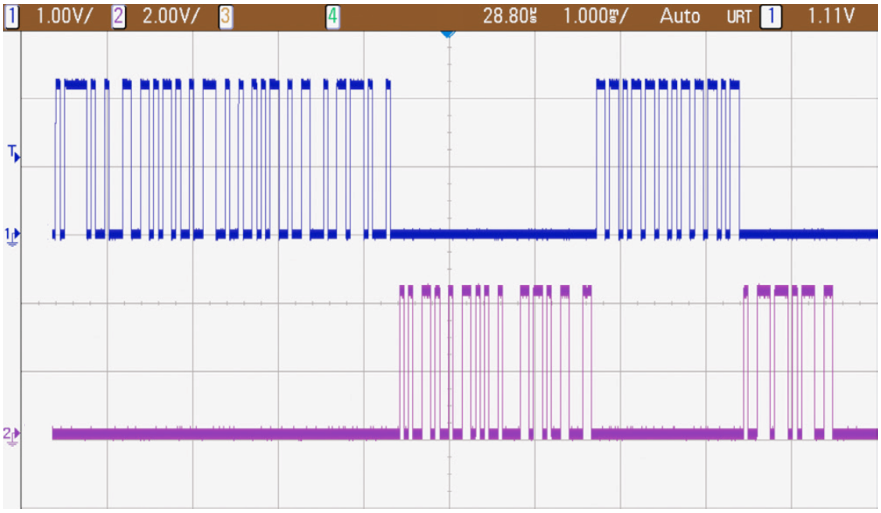
1.8) If a UART is set to a desired baud rate of 9600, what is the minimum clock frequency needed by the UART peripheral module?

- 9.6 kHz
- 86.4 kHz
- 96.0 kHz
- 105.6 kHz
- 153.6 kHz

1.9) When an IRQ is received by the NVIC which of the following does **NOT** happen?

- The state of the Core is saved to the stack
- The current instruction is completed before entering the ISR
- The current Stack Pointer is stored to the vector table
- The IPSR is set to the interrupt number
- The PC is set to the address of the ISR

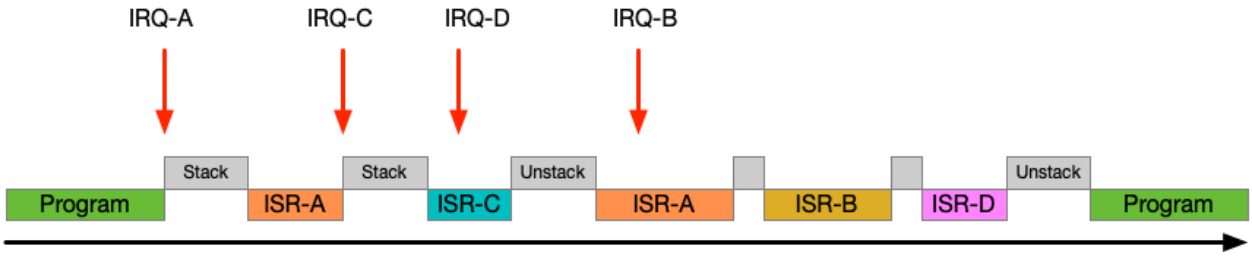
1.10) Which serial protocol is depicted in the oscilloscope screen shot below?



- Universal Asynchronous Receiver-Transmitter (UART)
- The Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I2C)
- Controller Area Network (CAN Bus)

Question 2: Interrupts

The interrupt timing diagram below shows four Interrupt Requests (IRQs) resulting in the execution of four ISRs. Use this diagram for the following questions.



2.1) Why should the time to execute an ISR be as short as possible? (choose the best answer)

- Short interrupts greatly reduce latency for the user
- Long interrupts can prevent other interrupts from running
- It does not matter ISRs only take 12 cycles to execute
- The shorter the code in the ISR the less time it takes for Stacking
- ISRs consume much more power then normal code execution

2.2) Assuming that the system is using PRIGROUP 0b101, assign a ***4-bit binary*** priority value to each of the interrupts shown in the figure above.

Table 51. Priority grouping

PRIGROUP [2:0]	Interrupt priority level value, PRI_N[7:4]			Number of	
	Binary point[1]	Group priority bits	Subpriority bits	Group priorities	Sub priorities
0b0xx	0bxxxx	[7:4]	None	16	None
0b100	0bxxx.y	[7:5]	[4]	8	2
0b101	0bxx.yy	[7:6]	[5:4]	4	4
0b110	0bx.yyy	[7]	[6:4]	2	8
0b111	0b.yyyy	None	[7:4]	None	16

1. PRI_n[7:4] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit.

A priority= __0b0100__or__0b0100_____(4-digit binary)
B priority= __0b1000__or__0b0101_____(4-digit binary)
C priority= __0b0000__or__0b0000_____(4-digit binary)
D priority= __0b1100__or__0b0110_____(4-digit binary)
**Multiple solutions exist these are examples

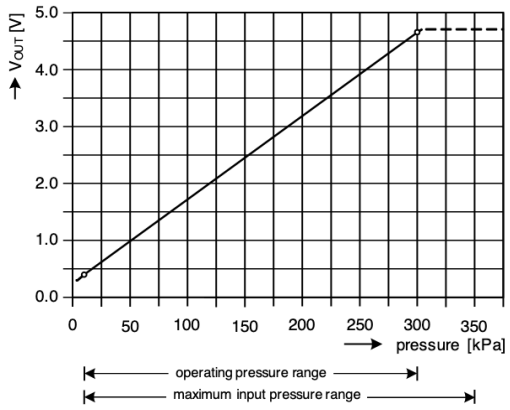
2.3) The ARM Cortex M microcontrollers have a number of methods for dealing with concurrent and overlapping interrupts. When the core switches from ISR-A to ISR-B this is an example of what type of method?

- Premptions
- Early Arrivals
- Later Arrivals
- Tail Chaining
- Fast Chaining

Question 3: ADC Sensor Measurement

You landed your first internship at NASA and you are asked to integrate an air pressure sensor into new weather balloons that measure atmospheric conditions before launch. The KP229E2701 is a miniaturized Analog Manifold Air Pressure Sensor that converts a pressure into an analog output signal. The calibrated transfer function (shown below) converts a pressure of 10 kPa to 300 kPa into a voltage range of 0.40 V to 4.65 V. You are given a 10 Bit ADC running off of a 5 volt power supply.

The transfer function is VOUT = Vref+ x (0.00293 x P_kPa + 0.05069)



3.1) What is the size of the Least Significant Bit (LSB) the ADC can measure in volts?

LSB = $(V_{Ref+} - V_{Ref-})/2^n = (5 - 0)/2^{10} = 4.88mV$

3.2) If the pressure is 100kPa what will the binary output value of the ADC be?

Output voltage of transducer

$V_{OUT} = V_{ref+} \times (0.00293 \times P\_kPa + 0.05069)$

$1.71845v = V_{ref+} \times (0.00293 \times 100 + 0.05069)$

Quantization error

± LSB = ±2.44mV

ADC Conversion Value

(1.71845v) * 1024 / 5v = 351.9 ≈ 352

Double Check

Steps * LSB + Quantization_Error = bit range

352 * 4.88mV + 2.44mV = 1.7202

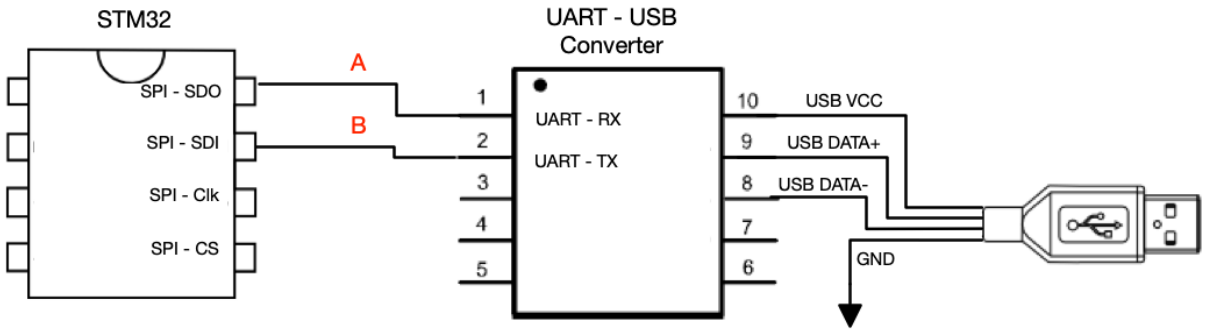
352 * 4.88mV - 2.44mV = 1.7153

The value recorded by the ADC will 352 (0b101100000 or 0x160)

Question 4: Serial Interfaces

An Ohio State CE student is trying to connect a STM32L4 MCU to a computer via Serial to USB converter module and needs your help debugging. He boasts that he is “a natural” at embedded systems and never needs to read the datasheet. He says he is getting strange results and the problem is probably a defective STM32L4. When he sends ASCII characters from the MCU to the Virtual Com Port on the computer the characters are wrong and sometimes throw an error. When he sends data from the computer’s Virtual Com Port to the MCU he mostly receives packets of all “1s” and only occasionally receives packets that contain a “0”.

The following figure shows the problematic hardware setup.



After examining the dev kits schematics you see that the student is using a UART-to-USB Protocol Converter module which allows a MCU to communicate with the Virtual COM Port on the host computer. After checking the datasheet you quickly verify that the UART-to-USB module is set up correctly and communicating with the virtual com port using a baud rate of 9600, one start bit, eight data bits, no parity bit, and one stop bit. Data is sent MSB first.

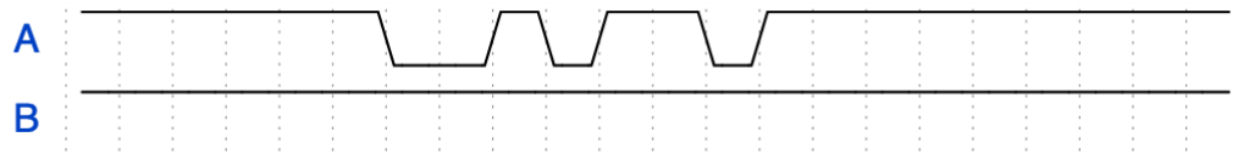
4.1) What is wrong with this approach?

The Ohio State student is trying to use SPI to communicate with UART device which are two different protocols with different bit encodings.

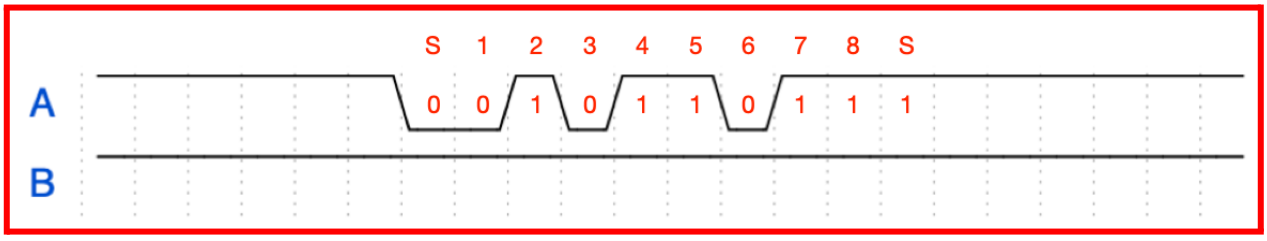
4.2) How is it possible that any data could be received on the computer? Hint: what can you say about the baud rate of the “UART to USB Converter” and the serial data clock on the STM32?

The baud rate of the UART to USB converter and serial data clock on the STM32 match (or is very close) so the bit length of the data being sent to the UART is of approximately the correct timing. Since the UART is an asynchronous protocol it does not need a clock to decode data. Even Though the data is SPI the UART on the “UART to USB Converter” chip still attempts to decode the packet by looking for the start bit followed by data bits, a parity bit, and a stop bit.

4.3) The Ohio State student shows you the following logic analyzer plot of the “A” and “B” lines as denoted in the above figure. What is the ASCII character that will be received by the virtual com port? (report the binary value for partial credit)



Answer: see the following diagram for the bit decoding



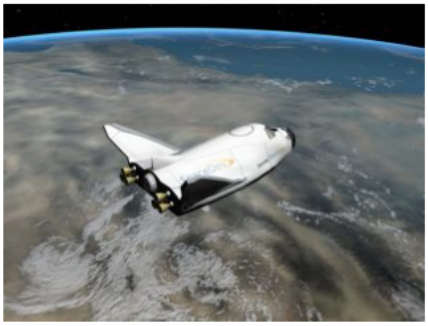
01011011 -> [
left square bracket

Question 5: Design Problem [50 points]

Background

After a successful internship at NASA, you have landed your first full time job at *SmithX Aerospace Company*. They are working on a reentry module (RM) that uses lifting body principles to slow its entry in the atmosphere like NASAs space shuttle.

The body is designed to have drag characteristics to slow its transition through the atmosphere. Simulations of the RM show it approaching a **terminal velocity of 250km/hr**. This is when the force of gravity is equal to the force from drag yielding a *constant velocity and acceleration approaching zero.* The aerodynamicists are interested to see if the altitude at which the RM reaches terminal velocity matches the simulation. You will be working with a team to instrument an RM prototype for a drop test from an aircraft.



The Components

At your disposal you will have the MAT32 processor, an I2C accelerometer and radar altimeter. The MAT32 processor has GPIO, I2C and timer peripherals. The I2C accelerometer is similar to the one used in the lab including the ability to generate interrupts at preset acceleration settings. The altimeter works by sending a pulse and measuring the time for a reflected pulse to return. The specifications for the processor and devices are in a separate document.

Measurement Strategy

1. To conserve battery life, the altimeter will not be activated until the terminal velocity is achieved.

2. To determine that terminal velocity has been achieved, set the accelerometer to interrupt when the acceleration is below **0.1 m/s2.** The accelerometer is oriented in the module to measure acceleration in the Z-axis. You need only measure this axis.

3. To get an accurate measurement immediately measure the altitude and use the timer capture feature. The timer capture only captures on a rising edge so you will have to zero the timer counter when you start the altimeter pulse.

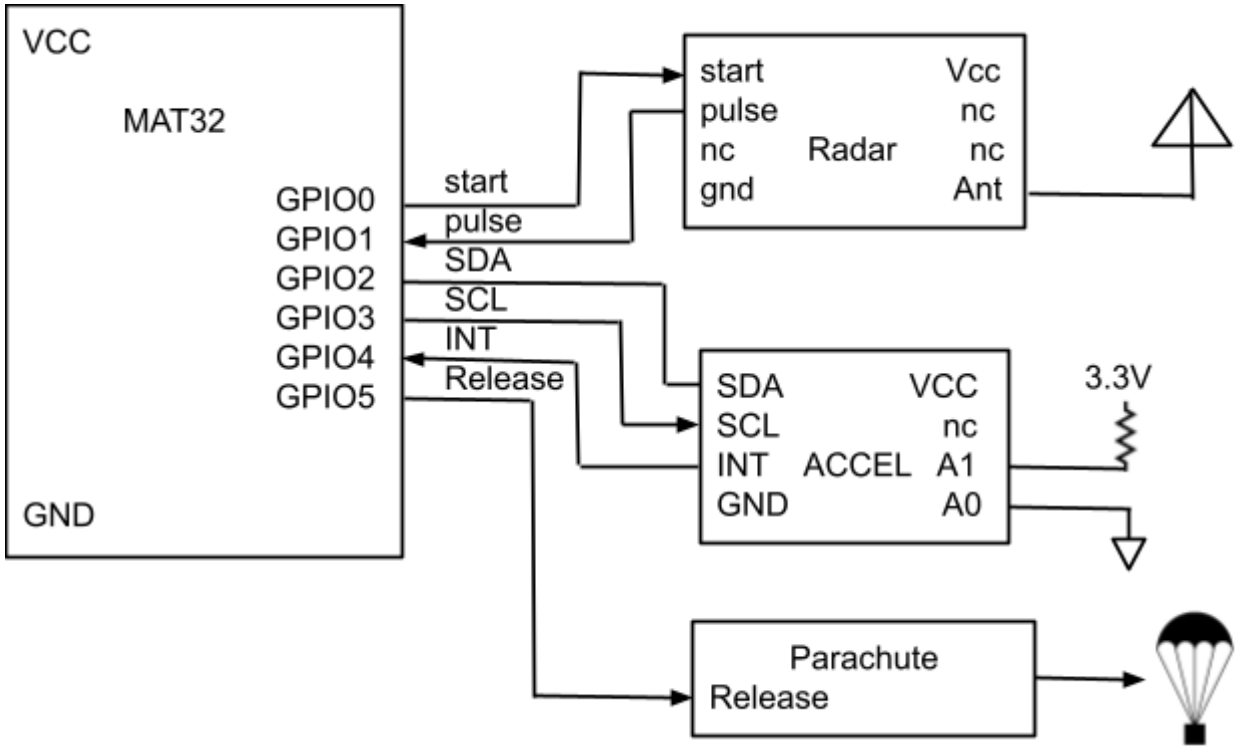
4. The threshold velocity altitude will be saved in a global variable.

5. Start taking altitude measurements **10 times a second** using one of the timers.

6. When the altitude is below **150 meters**, deploy the parachute.

System Diagram

The figure depicts the system diagram showing the connections between the components. Signals not required for this problem are omitted for clarity.



Part 1: Timer Setup (fill in the blank) (10)

Two timers will be required for the application: one timer to generate a 10Hz periodic interrupt for sampling the altimeter and another timer to measure the return time of the altimeter pulse.

TIM0

This timer will be used to initiate an altitude measurement 10 times a second. What values will you need for the prescaler, compare period and compare duty cycle to provide a *10Hz interrupt*. You can assume the *duty cycle is 50%* and the timer *source clock is 10MHz*. Set the timer to be *accurate to 1ms*

Prescaler_____ 9999 (provides accuracy of 1ms) other combinations with more accuracy were considered if they fit in a 16bit register.

ARR_____ 99 (count is zero referenced)

Compare/Capture_____ Depends on polarity and but either 49 or 50 because the application is not dependent on duty cycle.

TIM1

This timer will be used to measure the radar altimeter pulse travel time. It needs to measure an altitude as large as 1000 meters with an accuracy of *0.1 meters*. The *source clock is 10 MHz.* The pulse travels at 0.1 meters/micro second.

Prescaler_____ 9 /*provides 0.1 meter resolution

ARR_____ dc /*don’t care

Part 2: GPIO Setup (6)

Complete the code to initialize the GPIO Port using the “MAT32 - Firmware: GPIO” firmware drivers provided in the Exam datasheet.

Copy the following code and paste it into the answer field below. Replace the blank spaces “__” with the correct values. ½ point for each correct answer.

```
MAT_GPIO_Init(GPIO_0,__, __);   /*used to initiate start signal
MAT_GPIO_Init(GPIO_1,__, __);   /*timer 1 capture
MAT_GPIO_Init(GPIO_2,__, __);   /*SDA
MAT_GPIO_Init(GPIO_3,__, __);   /*SCL
MAT_GPIO_Init(GPIO_4,__, __);   /*interrupt
MAT_GPIO_Init(GPIO_5,__, __);   /*used release


MAT_GPIO_Init(GPIO_0,OUTPUT, 0);          /*used to initiate start
signal
MAT_GPIO_Init(GPIO_1,INPUT, AF1);     /*timer 1 capture
MAT_GPIO_Init(GPIO_2,INPUT, AF2);     /*SDA
MAT_GPIO_Init(GPIO_3,INPUT, AF2);     /*SCL
MAT_GPIO_Init(GPIO_4,RISING_INT, 0);  /*interrupt
MAT_GPIO_Init(GPIO_5,OUTPUT, 0);          /*used release

345#define AF0  1
#define AF1     2
#define AF2     4
```

Part 3: Accelerometer Setup (12)

Write the code to initialize the accelerometer to interrupt when the terminal velocity is met.

```
// Steps
// 1) Setup accelerometer use z axis with control register 0.
// 2) Setup accelerometer full scale value (any will work) with control register 0
// 3) Set accelerometer to interrupt on high to low transition thru threshold with control register 1
// 4) Enable z axis with threshold interrupt with control register 1
// 5) Write control registers 0 and 1 with I2C
// 6) Set interrupt threshold register value based on full scale setting and 0.1m/s^2
        // 11, +/-10m/s2, 0.1/(20/256)= 1.28, 1 or 2
        // 10, +/-5m/s2, 0.1/(10/256)= 2.56, 2 or 3
        // 01, +/-2m/s2, 0.1/(4/256)= 6.4, 6 or 7
        // 00, +/-1m/s2, 0.1/(2/256)= 12.8, 12 or 13
//7) Write z axis threshold register with I2C
```

// 1) Setup accelerometer use z axis with control register 0.
// 2) Setup accelerometer full scale value (any will work) with control register 0

```
//PZ PY PX SCL[1:0] EZ EY EX
//pending bits should be cleared but are assumed reset to start
#define cnt0 0bxxx 00 100
#define cnt0add 0
```

// 3) Set accelerometer to interrupt on high to low transition thru threshold with control register 1
// 4) Enable z axis with threshold interrupt with control register 1

```
//TYP[1:0] TCZ TCY TCX IEZ IEY IEX
#define cnt1 0b10 100 100
#define cnt1add 1
TYP[1:0] TCZ TCY TCX IEZ IEY IEX
```

// 5) Write control registers 0 and 1 with I2C

```
#define DevAddress 0b100 1010
uint8_t buf[3] = {cnt0add | 0x80, cnt0, cnt1};
MAT_I2C_Send(DevAddress<<1, buf, 3)
```

// 6) Set interrupt threshold register value based on full scale setting and 0.1m/s^2

```
#define z_thr = 12
#define cnt7add = 7
```

//7) Write z axis threshold register with I2C

```
uint8_t buf[2] = {cnt7add, z_thr};
MAT_I2C_Send(DevAddress<<1, buf, 2);
```

Part 4: Accelerometer Interrupt (14)

There are 3 interrupts for this application: the accelerometer interrupt, the capture interrupt and the 10Hz interrupt. We will ask you to write the accelerometer and capture interrupt. You can assume the 10Hz interrupt is written and starts the radar measurement with a start pulse and sets the capture timer counter to zero.

// Steps
// 1) Clear the accelerometer interrupt and disable.(may not be necessary)
// 2) Start the 10Hz altimeter timer 0.

// 3) Setup altimeter pulse capture timer 1.
// 4) Write the altimeter start signal and zero the timer1 counter to initiate immediate terminal
//velocity altitude measurement.

// 1) Clear the accelerometer interrupt and disable.(may not be necessary)

```
//clear pending Z threshold interrupt and disable accelerometer
#define DevAddress 0b1001010
#define cnt0 0b10000000
#define cnt0add 0
#define cnt1 0b00000000
#define cnt1add 1
uint8_t buf[3] = {cnt0add | 0x80, cnt0, cnt1};
int MAT_I2C_Send(DevAddress<<1, buf, 3)
```

// 2) Start the 10Hz altimeter timer 0.

```
#define TIM0 0x40001000
#define CNT 0
#define PSC 4
#define ARR 8
#define CRR 0xC
#define CNTR 0x10

uint32_t * timer0 = (uin32t *)(TIM0);
*(timer0+1) = 9999;
*(timer0+2) =  99;
*(timer0+3) = 49;
*(timer0+4) = 0b0101;//compare/capture,int en,polarity,PWMout dis/enb
```

// 3) Setup altimeter pulse capture timer 1.

```
#define TIM0 0x40002000
uint32_t * timer1 = (uin32t *)(TIM1);
*(timer1+1) = 9;
//*(timer1+2) = ;
//*(timer1+3) = ;
*(timer1+4) = 0b11x1;//compare/capture,int en,polarity,PWMout
```

// 4) Write the altimeter start signal and zero the timer1 counter to initiate immediate
// terminal velocity altitude measurement

```
// send altimeter pulse
MAT_GPIO_Write(GPIO_0, 1);
MAT_GPIO_Write(GPIO_0, 0);
// start pulse width measurement at zero
```

```
*TIM1 = 0;
```

Part 5: Capture Interrupt (8)

Write the capture interrupt. On the first occurrence, save the altitude to the global variable:

```
global float tv_altitude
```

// Steps
// 1) Get pulse measurement from timer 1 capture register.
// 2) Convert pulse measurement to meters.
// 3) Store measurement if it is the first capture interrupt.
// 4) Release parachute if altitude is less than 150 meters.

// 1) Get pulse measurement from timer 1 capture register.

```
#define TIM1 0x40002000
#define CCR 0x0c
unint32_t * timer1 = (unint32_t *)(TIM1);
uint32_t capture = * (timer1+3);
```

// 2) Convert pulse measurement to meters.

```
/* capture value is in us
/* altimeter provides 0.1meters/us
float altitude = ((float)capture*0.1)/2.0;
```

// 3) Store measurement if it is the first capture interrupt.

```
static uint32_t first_time = 0;
if (first_time)
     tv_altitude = altitude;
```

// 4) Release parachute if altitude is less than 150 meters.

```
if( altitude < 150.0)
     MAT_GPIO_Write(GPIO_5, 1);
```